Comment allons-nous développer d'ici 5 à 10 ans ?

Didier Vojtisek

▶ **To cite this version:**

Didier Vojtisek. Comment allons-nous développer d'ici 5 à 10 ans ?. Programmez !, 2012, 150, pp.44-46. hal-00714953

HAL Id: hal-00714953
https://inria.hal.science/hal-00714953

Submitted on 6 Jul 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Comment allons-nous développer d'ici 5 à 10 ans ?

Soucieux d'améliorer la qualité de son travail, le développeur cherche inlassablement les nouvelles technologies et les méthodes qui vont l'aider dans sa tâche. Certains outils actuels et les recherches dans le domaine du génie logiciel laissent présager certaines tendances pour les années à venir.

Au sein d'Inria, institut de recherche en informatique, les scientifiques sont amenés à réfléchir et à proposer des solutions à toutes sortes de problématiques dans différents domaines impliquant l'outil informatique. Ils développent des approches formelles ou pragmatiques pour améliorer des domaines très variés tels que l'internet, le réseau, le temps réel, la robotique, la santé (bio informatique) ou l'environnement.

Parmi les différentes équipes d'Inria, certaines (dont je fais partie depuis 11 ans) sont spécialisées dans les problématiques liées à la production logicielle. Elles collaborent avec des industriels et des PME pour trouver les moyens d'améliorer le développement des applications de demain tout en maitrisant la complexité et la qualité.

Qu'elles soient complexes ou simples, les applications doivent pouvoir fonctionner et s'adapter à différentes plateformes ou environnements. Il est fréquent que de nombreuses spécialités et métiers interagissent pour produire le logiciel qui devient le résultat de la réflexion commune des différents intervenants. Dans la pratique, nous constatons qu'il est difficile de trouver des spécialistes de tous ces domaines et de les faire communiquer.

Vers plus de capitalisation du savoir faire

Pour pallier à ces difficultés, une première grande tendance qui me semble prometteuse vise à une capitalisation plus importante des bonnes pratiques et des processus de développement. Pour preuve, regardez le nombre de sessions sur ALM (Application Lifecycle Management, gestion du cycle de vie applicatif) de la prochaine conférence EclipseCon.

En tant qu'informaticiens, nous allons évidemment nous appuyer des solutions qui vont nous permettre d'exploiter la machine afin de soulager le développeur des tâches fastidieuses et l'assister pour les tâches où il n'est pas spécialiste. De manière naïve, c'est ce que l'on faisait déjà en écrivant rapidement de petit scripts (ou « moulinettes ») ou en créant des assistants (wizards) générant la structure des tests ou encore une interface homme-machine minimale. Mais nous sommes aujourd'hui capables d'aller plus loin dans le guidage et l'automatisation du cycle de production. Par exemple, les technologies « Orientées par les Modèles » telles qu'UML, MDA, MDE et autre DSL (Domain Specific Language) offrent des techniques et des outils réutilisables d'un domaine à l'autre. Typiquement, on peut utiliser des modèles qui sont à la fois manipulables par la machine, qui offrent une certaine abstraction compréhensibles pour l'humain et qui lui masquent une partie de la complexité. La machine réalise alors les opérations complexes ou fastidieuses à la place de l'humain, et lui laisse alors plus de temps pour interagir avec le reste de l'équipe et se concentrer sur des tâches de plus hauts niveaux.

Rappelons-nous comme l'histoire de l'informatique nous a déjà habitués à des changements de niveau d'abstraction pour répondre à la complexité croissante des applications. Une succession de technologies proposant des abstractions de plus en plus élevées sont arrivées : les langages assembleurs, puis des langages procéduraux comme le C, puis les langages orientés objets complétés par des frameworks de plus en plus évolués, et aujourd'hui les langages orientés modèles comme UML ou des DSL. A chaque étape, les développeurs ont progressivement adopté la nouvelle abstraction la mieux adaptée à l'expression de leur métier, laissant aux spécialistes des langages de plus bas niveau le soin de mettre leur savoir faire dans les compilateurs. Pendant les phases de transitions et en attendant que les compilateurs soient suffisamment performants, nombreux ont été ceux qui ont mixés les langages pour y suppléer et optimiser le code résultant. Aujourd'hui, les technologies modèles nécessitent encore d'avoir besoin de contrôler ou d'adapter le code produit, pourtant déjà dans certains domaines métier les outils fournissent des transformations et des générateurs ne nécessitant que peu de retouches. On peut donc s'attendre à une généralisation de cette tendance. D'autant plus, que l'on note actuellement une convergence entre le modèle et le code. En effet, de nombreux modèles disposent d'une syntaxe textuelle car certains concepts se représentent parfois mieux sous forme textuelle que sous forme graphique avec des diagrammes. Ces modèles s'apparentent alors à du code.

Plus globalement dans le domaine de l'amélioration des processus de conception, on tend vers un meilleur découplage des préoccupations (en anglais : *separation of concerns*) qui consiste à découper le programme (ou modèle) en aspects aussi indépendants que possibles les uns des autres. Grâce à des vues limitées à un sous ensemble des aspects de l'application, chaque métier peut s'exprimer de manière plus naturelle puis intégrer son travail dans la réalisation finale. Ainsi on essaiera de laisser la définition des exigences fonctionnelles au client ou bien laisser la spécification des aspects sécurité, distribution, stockage en base de données et tolérance aux fautes à des experts de ces différents domaines. L'activité classique du développeur consiste alors à assembler manuellement ces préoccupations dans le code final, (Figure 1) ce qui peut être fastidieux et source d'erreur, surtout lors des évolutions successives du logiciel.

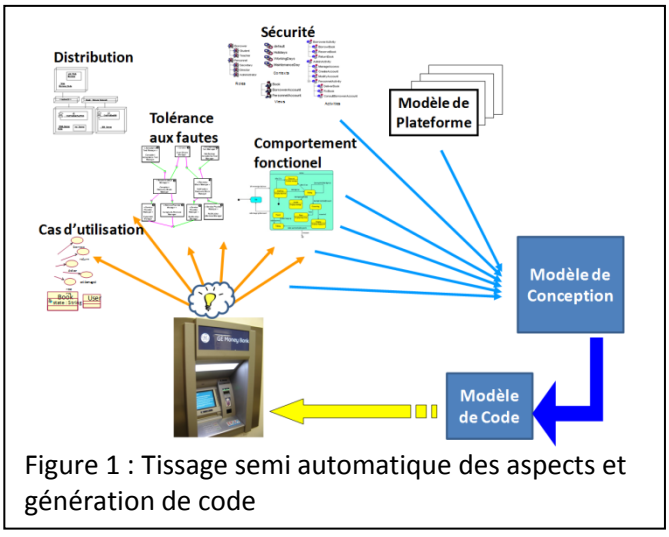

Figure 1 : Tissage semi automatique des aspects et génération de code

Pour simplifier l'assemblage de ces préoccupations, les techniques orientées aspects apportent des solutions et devraient connaitre un certain essor dans les années à venir. En effet, les préoccupations peuvent être intégrées à plusieurs niveaux du cycle de production.

Au niveau code, on utilisera probablement les capacités aspect de langages comme aspectJ ou les capacités de réouverture de classe de langages comme scala, Ruby ou C# pour injecter et assembler les préoccupations. De plus, les techniques d'injection par annotation que l'on trouve dans des langages comme Java et qui sont largement utilisé par des framework comme Spring permettent aussi de bien séparer le code fonctionnel du code technique lié à des préoccupations transverses.

L'approche MDE propose aussi d'appliquer ces pratiques au niveau des autres étapes de la conception telles que l'expression des besoins, l'analyse, la conception ou les tests. En exploitant des modèles pour représenter ces étapes, il devient possible d'automatiser partiellement certaines transitions. Ceci permet de raffiner progressivement le modèle de conception en son code final. De manière pragmatique, il est normal que l'automatisation ne soit jamais complète. L'arbitrage d'un humain reste indispensable et on peut s'attendre à ce qu'il agisse sur le choix des outils, sur leur paramétrage ou bien adapte et complète directement les modèles ou codes produits par les outils.

Les électroniciens qui conçoivent des cartes sont très demandeurs de cette approche de raffinage successif. Les systèmes modernes sont si complexes à produire qu'ils doivent être capables de concevoir certaines étapes en faisant temporairement abstraction de certaines problématiques puis de raffiner progressivement le design final. Ils pourront s'abstraire de notions telles que l'horloge ou le placement lors de la phase de spécification des besoins fonctionnels.

D'autres éléments montrent que l'on capitalise de plus en plus sur les processus. Si l'on considère le flot de conception dans son ensemble, des modèles exprimés en BPMN (Business Process Modeling Notation) ou en SPEM (Software Process Engineering Metamodel) sont capables de modéliser les différentes étapes de conceptions. Ces étapes peuvent représenter toutes sortes de processus de développement comme les processus classiques en V ou les méthodes agiles. Là encore, les technologies et les outils sont jeunes et il faudra les mettre à l'épreuve pour les adapter aux méthodes concrètement suivies en entreprises. Ces modèles de processus sont particulièrement intéressants car ils permettent de communiquer sur les choix d'outils sur tout le cycle de vie du logiciel. Pour l'instant, il revient à chaque entreprise de capitaliser son processus de développement interne, mais nul doute qu'émergeront des processus réutilisables que l'on pourra appliquer ou combiner pour nos futurs développements.

L'utilisation des différentes méthodes évoquées ci-dessus apportent des garanties de reproductibilité des développements et améliorent leur qualité. Certaines études récentes montrent que malgré un coût de formation initial, le MDE peut être un allié de choix pour les méthodes agiles. Sur l'introduction d'un nouveau besoin, le MDE offre une bonne réactivité en donnant la possibilité d'adapter le processus et de régénérer directement ce qui peut l'être. Il permettra aussi d'adresser plus facilement différentes cibles en spécifiant la variabilité puis en sélectionnant les fonctionnalités spécifiques à chaque version. Cela crée des lignes de produit (*Software Product Line*) analogues à ce que l'on peut faire dans d'autres industries.

Vers plus de fiabilité

Une seconde grande tendance qui a commencé à se dessiner pour le développement est le support des systèmes dynamiques capables non seulement de prendre en compte les différentes plateformes d'exécution mais aussi de s'adapter à l'exécution *(@runtime*).

En utilisant les méthodes orientées modèles citées plus haut, il est déjà possible de prendre en compte la variabilité de l'application dans les phases de développement *(@designTime*). Le concepteur qui choisira soigneusement ses outils pourra ainsi créer différentes variantes de son logiciel, une version web, une version Android, une version linux,… Il pourra aussi intégrer des choix fonctionnels dans son application par exemple en sélectionnant le support de tel ou tel protocole.

Pourtant les objets utilisant informatiques tendent à changer : grâce une interconnexion croissante et la baisse du coût des équipements, de plus en plus de d'objets deviennent intelligents. Nous constatons aussi que les utilisateurs s'attendent de manière croissante à avoir les services disponibles en permanences.

L'un des enjeux actuels est d'arriver à construire des systèmes qui ne s'arrêtent jamais, même si les conditions d'utilisation changent. Cela s'applique à tous les types d'équipements, des microcontrôleurs embarqués jusqu'aux serveurs et systèmes d'information. C'est déjà une nécessité au niveau des clouds, datacenters et autre application centers. Ils offrent pour la plupart des mécanismes d'adaptation à la charge de travail. Souvent basées sur des approches orientées composant, certaines solutions applicables à des plateformes logicielles offrent déjà des mécanismes aidant à la conception de systèmes capables d'échanger un composant logiciel par une version plus récente ou par une version offrant un service différent. Certaines de ces plateformes offrent même des moteurs d'analyse permettant de s'adapter automatiquement à des stimuli pour reconfigurer le système. Ces stimuli peuvent être très variés : une réaction à une panne, à une diminution de la qualité de service de l'un des composants ou bien même l'intégration d'un nouveau besoin fonctionnel. La difficulté dans ce cas est la gestion fiable des cas non planifiés et donc de gérer le nombre élevé de configurations possibles car il est quasi impossible de toutes les tester lors de la conception. Il existe aussi des expérimentations montrant des reconfigurations de très bas niveau, par exemple en reconfigurant à chaud un microcontrôleur. Il est même possible de reconfigurer du matériel en reprogrammant un FPGA pour qu'il soit capable de traiter un algorithme différent afin de passer du traitement d'un codage vidéo à un autre ou pour traiter un signal radio différent.

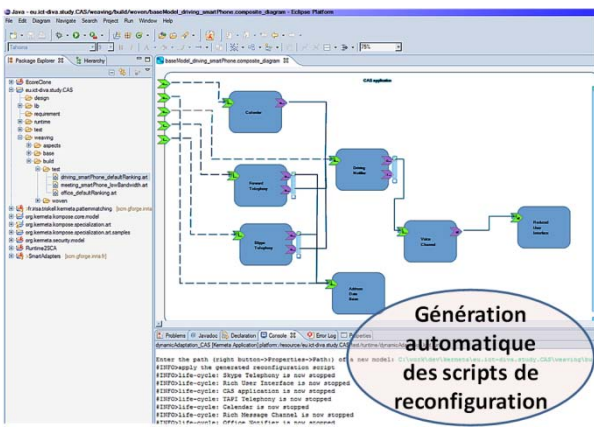

Figure 2 : Reconfiguration dynamique des fonctions d'un téléphone pour s'adapter à une nouvelle situation.

Conclusion

L'évolution technologique ne cesse de s'accélérer, avec aujourd'hui des architectures potentielles multiples, et souvent des besoins hybrides suivant les différentes parties de l'application. Pour satisfaire les utilisateurs, les applications requièrent d'être aussi fiables que possible en toutes circonstances. Différentes pratiques permettent de mieux maitriser la complexité croissante induite par ces besoins. Parmi elles, les technologies orientées modèles me semblent très prometteuses car non seulement elles unifient des solutions existantes, mais elles offrent aussi des réponses innovantes aux problématiques du développement des applications modernes. Certaines étapes de développement nécessiteront encore d'être affinées, comme l'expression des besoins qui n'a pas encore trouvé de formalisme faisant l'unanimité. Néanmoins, les gains étant particulièrement significatifs sur le long terme pour la maintenance et l'évolution, la plupart des développeurs ont tout intérêt à s'investir dans ce genre de technologies. En tant qu'utilisateurs des outils, ils pourront ainsi profiter des avancées régulières. Mieux, certains pourront capitaliser une partie de leur savoir faire pour le retransmettre à leurs pairs.

Aujourd'hui la conception des processus reste encore l'affaire de spécialistes qui analysent notre manière de travailler. A terme, lorsque les processus de développement seront mieux spécifiés et outillés de manière modulaire, chaque développeur pourra être l'architecte et le coordinateur de sa propre méthode de travail s'il le désire.

Didier Vojtisek
Ingénieur de recherche Inria
didier.vojtisek@inria.fr